DBSVEC: Density-Based Clustering Using Support Vector Expansion

Zhen Wang[†,‡], Rui Zhang[‡,*], Jianzhong Qi[‡], Bo Yuan[†]
[†]Graduate School at Shenzhen, Tsinghua University, China
[‡]School of Computing and Information Systems, The University of Melbourne, Australia
z-wang16@mails.tsinghua.edu.cn rui.zhang@unimelb.edu.au
jianzhong.qi@unimelb.edu.au yuanb@sz.tsinghua.edu.cn

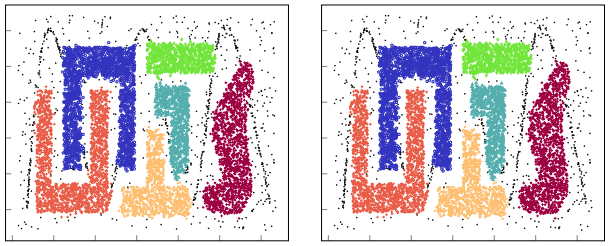
***Abstract*—DBSCAN is a popular clustering algorithm that can discover clusters of arbitrary shapes with broad applications. However, DBSCAN is computationally expensive, as it performs range queries for all the points to determine their neighbors and grow the clusters. To address this problem, we propose a novel approximate density-based clustering algorithm named DBSVEC. DBSVEC introduces *support vectors* into density-based clustering, which allows performing range queries only on a small subset of points called the *core support vectors*. This technique significantly improves the efficiency while retaining high-quality cluster results. We evaluate the performance of DBSVEC via extensive experiments on real and synthetic datasets. The results show that DBSVEC is up to three orders of magnitude faster than DBSCAN. Compared with the state-of-the-art approximate density-based clustering methods, DBSVEC is up to two orders of magnitude faster, and the clustering results of DBSVEC are more similar to those of DBSCAN.**

***Index Terms*—density-based clustering, support vector expansion, scalable clustering**

I. Introduction

Clustering is a fundamental problem in data mining, and the density-based clustering algorithm DBSCAN [1] is one of the most influential techniques due to its capability to find clusters of arbitrary shapes. It has broad applications in many fields such as spatial data analysis [2], science of astronomy [3], and biomedical research [4]. DBSCAN connects contiguous *core points*, separated by regions of low point-density, to form clusters. A core point is a point that has at least $MinPts$ points around it within an ϵ-radius sphere, where $MinPts$ and ϵ are user-defined parameters. Although highly effective, DBSCAN suffers from efficiency issues especially when dealing with large scale datasets. This is due to the fact that, when connecting contiguous core points (i.e., cluster expansion), DBSCAN requires running range queries for each data point to test whether it satisfies the core point criteria. It has been shown [5] that even with speedup indexing techniques such as kd-trees [6] or R-trees [7], the worst-case time complexity of DBSCAN is still $O(n^2)$, where n is the number of points in a dataset.

Approximate DBSCAN algorithms have drawn significant attention of the community. Such algorithms speed up DBSCAN using *approximate range queries* such as hierarchical grid structures [5], [8], [9] and *Locality Sensitive Hashing* (LSH) [10], [11]. Grid structures can be used to replace range queries with simple counts over the number of points in the neighboring grid cells (i.e., grid cells in the query range). However, the performance of grid-based approximation algorithms is heavily impacted by data dimensionality d: the number of grid cells increases rapidly with d. For example, the ρ-Approximate DBSCAN algorithm [5] is grid-based and has time complexity of $O\left(n(\frac{1}{\rho})^d\right)$, which increases exponentially with d (ρ is a system parameter with small values such as 0.001 by default). The use of LSH [12] can help reduce data dimensionality by hashing data points from the original high dimensional space to a much lower dimensional space. Unfortunately, such data dimensionality reduction causes the loss of accuracy of the clustering results. Also, a high space cost may be incurred to maintain the hash table.



(a) DBSCAN on *t4.8k* (b) DBSVEC on *t4.8k*

Fig. 1: Clustering quality of DBSVEC

We aim to produce the same clustering output as DBSCAN does, but in a much more efficient way. We propose an algorithm named ***D**ensity-**B**ased **S**upport **V**ector **E**xpansion **C**lustering* (DBSVEC) to achieve this goal. Figure 1 compares the clusters produced by DBSVEC and DBSCAN on a public dataset *t4.8k* [13] where each color represents a cluster. As the figure shows, the clusters produced by the two algorithms are the same. Meanwhile, DBSVEC is 7.7 times faster than DBSCAN on this dataset.

Our algorithm is based on the key insight that, after an initial cluster has been identified, we only need to run range queries for points on the boundary of the current cluster (rather than every point in the cluster) to expand it. This helps to avoid unnecessary range queries and hence reduce the running time substantially.

Specifically, we identify a small number of points around the boundary of an expanding cluster such that their ϵ-neighborhood (the set of points within distance ϵ from a point) together can approximately cover the same set of new points

* Corresponding author.
This work was done when Zhen Wang was visiting University of Melbourne

as those covered by the ϵ-neighborhood of all the points in the cluster. To identify those boundary points, we exploit *Support Vector Domain Description* (SVDD) [14], a technique that finds a set of support vectors (points) describing a *closed* boundary of a set of points. Computing SVDD on the set of points in an expanding cluster produces support vectors on the boundary of the cluster. We show that a small constant number of support vectors are sufficient to cover most of the ϵ-neighborhood of the expanding cluster. Thus, we can obtain clusters similar to those of DBSCAN (i.e., ensure high clustering accuracy) but with much lower computational cost.

Furthermore, we propose three techniques to improve and speed up the iterative computation of SVDD in DBSVEC, so that DBSVEC becomes more accurate and efficient. First, we propose a variant of the SVDD model with an adaptive penalty weight for each data point. This weight guides the support vector computation process towards selecting points on the boundary of an expanding cluster. Performing range queries on such points helps to obtain more similar clustering results to those of DBSCAN. Second, we propose an incremental learning method that enables SVDD to focus on newly added points rather than retesting the whole set of points in a cluster. This reduces the computation cost. Third, we present a kernel parameter value selection strategy to alleviate model overfitting and hence to avoid generating an excessive number of support vectors for running range queries on.

In summary, this paper makes the following contributions:

- We propose a highly efficient density-based clustering algorithm named DBSVEC for very large datasets. This is the first work that exploits support vectors to reduce the number of unnecessary range queries in DBSCAN. It significantly improves the efficiency of DBSCAN while retaining high clustering accuracy. We also show that, only under very strict conditions, the clustering result of DBSVEC may deviates from that of DBSCAN.
- We propose three techniques to enhance SVDD for both effective and efficient clustering: (i) an adaptively weighted SVDD model, which assigns an adaptive penalty weight to each data point based on its position and the number of times participating in support vector computation, and hence improves clustering accuracy; (ii) an incremental learning method to further improve the efficiency of our SVDD algorithm, which allows a linear time complexity for support vector computation; (iii) a kernel parameter value selection strategy to alleviate model overfitting and hence to avoid generating an excessive number of support vectors.
- We perform an extensive experimental study on both real and synthetic datasets, which show that DBSVEC is up to three orders of magnitude faster than DBSCAN. Compared with state-of-the-art approximate density-based clustering methods, DBSVEC is up to two orders of magnitude faster, and the clustering results of DBSVEC are more similar to those of DBSCAN.

II. Preliminaries and Related Work

We first review density-based clustering, DBSCAN, and approximate density-based clustering algorithms. Then, we briefly discuss SVDD, a key component of our proposed algorithm. The frequently used symbols are listed in Table I.

TABLE I: Frequently Used Symbols

Symbol	Description
$\mathbf{X}$	A data set
$\mathbf{x}_i$	A data point
$n = \lvert\mathbf{X}\rvert$	The cardinality of $\mathbf{X}$
d	The dimensionality of $\mathbf{X}$
$MinPts$	Density threshold
ϵ	Radius parameter of clustering
$\mathcal{N}_\epsilon(\mathbf{x}_i)$	The ϵ-neighborhood of a point $\mathbf{x}_i$
Cl	A cluster
$\mathscr{S}$	A sub-cluster
$\widetilde{n} \leq \lvert\mathscr{S}\rvert$	Size of target data
R	Sphere radius
$\mathbf{a}$	Sphere center
ξ_i	Slack variable
C, ν	Penalty factors
α_i, β_i	Lagrange multipliers
Φ	Nonlinear function
K	Kernel function
σ	Kernel RMS width parameter
ω_i	penalty weight

A. Density-based Clustering

Let $\mathbf{X}$ be a set of n points in a d-dimensional space $\mathbb{R}^d$ where $\mathbf{x}_i = (x_{i1}, ..., x_{id})$ denotes the ith point and x_{ij} denotes its coordinate in the jth dimension.

We denote by $dist()$ the Euclidean distance function and assume two input parameters: radius $\epsilon \in \mathbb{R}^+$ and density threshold $MinPts \in \mathbb{N}^+$.

Definition 1 (ϵ − neighborhood): The ϵ-neighborhood of a point $\mathbf{x}_i$, denoted by $\mathcal{N}_\epsilon(\mathbf{x}_i)$, is the set of all points in a d-dimensional hypersphere centered at $\mathbf{x}_i$ with radius ϵ.

$$\mathcal{N}_\epsilon(\mathbf{x}_i) = \{\mathbf{x}_j \in \mathbf{X} | dist(\mathbf{x}_i, \mathbf{x}_j) \leq \epsilon\}$$

$\mathcal{N}_\epsilon(\mathbf{x}_i)$ is "dense" if it covers at least $MinPts$ points in $\mathbf{X}$. If $\mathcal{N}_\epsilon(\mathbf{x}_i)$ is dense, then $\mathbf{x}_i$ is called a *core point*.

Definition 2 (Core point): A point $\mathbf{x}_i \in \mathbf{X}$ is a core point if $\mathcal{N}_\epsilon(\mathbf{x}_i)$ contains at least $MinPts$ points in $\mathbf{X}$ (including $\mathbf{x}_i$ itself), i.e., $|\mathcal{N}_\epsilon(\mathbf{x}_i)| \geq MinPts$.

If a point is not a core point, it is called a *non-core point*. If the ϵ-neighborhood of a non-core point $\mathbf{x}_i$ contains at least one core point, $\mathbf{x}_i$ is called a *border point*; otherwise, $\mathbf{x}_i$ is called a *noise point*.

Definition 3 (Density-reachable): A point $\mathbf{x}_i$ is said to be density-reachable from a point $\mathbf{x}_j$ if there is a sequence of points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$ where $\mathbf{x}_1 = \mathbf{x}_j$ and $\mathbf{x}_t = \mathbf{x}_i$ such that $\mathbf{x}_1$, $\mathbf{x}_2, \ldots, \mathbf{x}_{t-1}$ are core points and $\mathbf{x}_{k+1} \in \mathcal{N}_\epsilon(\mathbf{x}_k)$ for $k \in [1, t\text{-}1]$.

Density-reachable is symmetric for two core points $\mathbf{x}_i$ and $\mathbf{x}_j$. On the other hand, points are not density-reachable from any non-core points.

Definition 4 (Cluster): A cluster Cl with respect to ϵ and $MinPts$ is a nonempty subset of $\mathbf{X}$ that satisfies:

- (Maximality) If a core point $\mathbf{x}_i \in Cl$, then all the points density-reachable from $\mathbf{x}_i$ also belong to Cl.
- (Connectivity) $\forall \mathbf{x}_i, \mathbf{x}_j \in Cl$, there is a core point $\mathbf{x}_k \in Cl$ such that both $\mathbf{x}_i$ and $\mathbf{x}_j$ are density-reachable from $\mathbf{x}_k$.

Algorithm 1 DBSCAN

Input: A finite set of points $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}^t$ in d-dimensional space $\mathbb{R}^d$ with $\mathbf{x}_i = (x_{i1}, ..., x_{id})^t$ being the ith point; a radius ϵ; a density threshold $MinPts$.
Output: Cluster ID of each point.

```
1: Cid ←0
2: for each unclassified point x_i ∈ X do
3:    N_ε(x_i) ←RangQuery(X, x_i, ε)
4:    if |N_ε(x_i)| ≥ MinPts then
5:       Cid ←Cid + 1;
6:       x_i.id ←Cid; S ←N_ε(x_i)
7:       for each unclassified point x_j ∈ S do
8:          N_ε(x_j) ←RangQuery(X, x_j, ε)
9:          if N_ε(x_j) ≥ MinPts then
10:            S ←N_ε(x_j) ∪ S
11:         if x_j does not belong to any cluster then
12:            x_j.id ←Cid
13:   else
14:      x_i.id ←noise
```

We study density-based clustering defined as follows.

Problem 1 (Density-based Clustering): Density-based clustering is to find the unique set $\mathscr{C}$ of clusters of $\mathbf{X}$.

B. DBSCAN

DBSCAN [1] (Algorithm 1) starts with an arbitrary point $\mathbf{x}_i$ and retrieves all points density-reachable from $\mathbf{x}_i$ (Lines 1 to 2). If $\mathbf{x}_i$ is a core point, a new cluster is identified (Lines 3 to 6, where $\mathbf{x}_i.id$ is used to store the cluster ID of $\mathbf{x}_i$ and Cid is a unique integer ID for different clusters). DBSCAN expands this cluster via repeatedly visiting the points in the cluster and adding their density-reachable neighbors into the cluster (Lines 7 to 12). The process continues until no new points can be added to this cluster. Then, a new unvisited point is selected, from which the above process resumes. If $\mathbf{x}_i$ is a non-core point, no points are density-reachable from $\mathbf{x}_i$ and DBSCAN moves onto the next arbitrary point that has not been visited yet (Lines 13 to 14). When all points have been visited, those points not in any cluster are regarded as noise.

DBSCAN requires $O(n^2)$ time [5]. Due to this high time complexity, many approximate algorithms have been proposed to improve the performance of DBSCAN.

C. Approximate Density-based Clustering Algorithms

We focus on approximate DBSCAN algorithms. Approximation techniques for other density-based clustering algorithms [15], [16] are less relevant and not discussed further.

Grid-based algorithms. The basic idea of grid-based algorithms is to divide the whole dataset into equal-sized square-shaped grids [5], [8], [9]. ρ-Approximate DBSCAN [5] is the state-of-the-art approximate implementation of DBSCAN. It uses a quadtree-like hierarchical grid with a cell width of $\epsilon\rho/\sqrt{d}$, where ρ is a system parameter to trade accuracy for efficiency. This grid is used to reduce the computational complexity of the range queries. Each range query now counts the number of points in $O\left(1 + (1/\rho)^{d-1}\right)$ cells. This algorithm has a linear time with regard to the dataset cardinality n in low-dimensional spaces (when $d \leq 7$). However, the number of grid cells accessed per range query increases exponentially with d. Recently, Schubert et al. [17] argue that the original DBSCAN algorithm with a proper configuration performs competitively with ρ-Approximate DBSCAN.

Hashing-based algorithms. Wu et al. [10] use *Locality Sensitive Hashing* (LSH) [12] to search for approximate nearest neighbor points to form clusters. DBSCAN-LSH [11] uses LSH for approximate distance computations, which reduces the number of distance computations for clustering. These algorithms lack formal analysis of the accuracy of the clustering results and may produce highly inaccurate clusters as shown in our experimental study (Section V).

Other fast algorithms. FDBSCAN [18] chooses the points that are far away from the core points to perform range queries, but it lacks accuracy analysis and experiments. Furthermore, FDBSCAN does not consider cluster expansion, which causes unnecessary range queries. NQ-DBSCAN [19] uses a local neighborhood searching technique for reducing the cost of distance computations. However, it does not reduce the number of range queries. P$^+$-tree [20] provides a way to accelerate nearest neighbor queries in high dimensional space, which divide the space into subspaces based on clustering so that the Pyramid technique can be applied in each subspace.

TABLE II: Complexity of Density-based Algorithms

Algorithm	DBSCAN [1]	ρ-Appr [5]	DBSCAN-LSH [21]	NQ-DBSCAN [19]	DBSVEC (this paper)
Complexity	$O(n^2)$	$O\left(n(\frac{1}{\rho})^d\right)$	$O(\ell n d^{k/2})$	$O(n^2)$	$O(\theta n)$

Table II summarizes the computational complexity of fast DBSCAN algorithms, where ℓ is the number of iterations, k is the number of hash functions, and $\theta \ll n$ is analyzed in detail in Section III-D. Unlike existing techniques which perform range queries on every data point, our algorithm DBSVEC introduces *support vectors*, which avoids unnecessary range queries with a small sacrifice in the clustering accuracy. DBSVEC is based on SVDD described below.

D. Support Vector Domain Description

Support Vector Domain Description (SVDD) [14] finds the minimum hypersphere that encloses all or most of the points in a dataset. The *hypersphere* (sphere for short hereafter) is defined by its radius R and center $\mathbf{a}$. Formally, SVDD computes the following optimization problem:

$$\begin{aligned} \min \quad & f(R, \mathbf{a}, \xi_i) = R^2 + C \sum_{i=1}^{n} \xi_i \\ s.t. \quad & \| \mathbf{x}_i - \mathbf{a} \|^2 \leq R^2 + \xi_i \\ & \xi_i \geq 0, \quad \forall i \end{aligned} \tag{1}$$

where C is a penalty factor that controls the trade-off between the two error terms: the volume of the sphere and the number of data points outside the sphere; ξ_i is the slack variable used to represent how far away the i-th point falls outside the sphere. The two constraints in Eq. 1 can be incorporated into the optimization function using Lagrange multipliers:

$$\begin{aligned} \mathcal{L}(R,\mathbf{a},\xi) = R^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i(R^2+\xi_i-\|\mathbf{x}_i-\mathbf{a}\|^2) - \sum_{i=1}^{n}\beta_i\xi_i \\ s.t. \quad \alpha_i \geq 0,\ \beta_i \geq 0 \end{aligned} \tag{2}$$

where α_i and β_i are Lagrange multipliers. By computing the partial derivatives of $\mathcal{L}$ in Eq. 2 with respect to R, $\mathbf{a}$ and ξ_i,

and letting them be 0, we obtain:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial R}=0: &\quad \sum_{i=1}^{n}\alpha_i=1\\ \frac{\partial \mathcal{L}}{\partial \mathbf{a}}=0: &\quad \mathbf{a}=\frac{\sum_{i=1}^{n}\alpha_i\mathbf{x}_i}{\sum_{i=1}^{n}\alpha_i}=\sum_{i=1}^{n}\alpha_i\mathbf{x}_i\\ \frac{\partial \mathcal{L}}{\partial \xi_i}=0: &\quad \alpha_i=C-\beta_i\end{aligned} \tag{3}$$

Substituting Eq. 3 into Eq. 1 results in:

$$\begin{aligned}\mathcal{L}=&\sum_{i=1}^{n}\alpha_i(\mathbf{x}_i\cdot\mathbf{x}_i)-\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j(\mathbf{x}_i\cdot\mathbf{x}_j)\\ s.t.\quad &0\le\alpha_i\le C,\quad \sum_{i=1}^{n}\alpha_i=1\end{aligned} \tag{4}$$

Maximizing Eq. 4 yields the value of α_i [14]. Only points $\mathbf{x}_i$ with $\alpha_i > 0$ are needed in the description of the sphere, and these points are called *support vectors* (SVs), which lie on the boundary of the sphere. SVs with $\alpha_i=C$, corresponding to points outside the sphere, are called *boundary support vectors* (BSVs). The SVs with $0<\alpha_i<C$, called *normal support vectors* (NSVs), correspond to points on the surface of the sphere.

III. Proposed Algorithm

We now present our algorithm DBSVEC. We start with the key idea of DBSVEC in Section III-A, followed by algorithm details in Section III-B. The algorithm accuracy and costs are analyzed in Sections III-C and III-D, respectively.

A. Key Idea

Our key idea is that many of the range queries in DBSCAN for core point tests are unnecessary and can be avoided.

Figure 2 illustrates a typical situation where some range queries can be safely removed, where the diamonds, triangles and dots all represent points to be clustered. During a DBSCAN run ($MinPts$=8 and the circles have radius ϵ), suppose $\mathbf{x}_1$ (the dot) is the first point visited. A range query finds that $\mathbf{x}_1$ is a core point, i.e., there are at least 8 ($MinPts$) points inside the solid circle centered at $\mathbf{x}_1$ (including $\mathbf{x}_1$ itself). The points in this circle form the ϵ-neighborhood of $\mathbf{x}_1$. DBSCAN then needs to run 7 range queries on the rest 7 points (denoted by the 7 circles centered at them) to expand the current cluster formed by this ϵ-neighborhood. However, many of these range queries overlap with each other and the sets of points they cover also overlap heavily (for example, the sets of points covered by the five dotted circles contain many identical points). A subset of these range queries (i.e., the range queries of $\mathbf{x}_2$ and $\mathbf{x}_3$ as represented by the other two solid circles) is sufficient to cover all the points enclosed by all the range queries. Consequently, the rest of the range queries (the dotted circles) are unnecessary.

We use the concept of *sub-cluster* to help identify the range queries necessary. A sub-cluster is a subset of a cluster that satisfies the connectivity but not the maximality requirement of a cluster (cf. Definition 4):

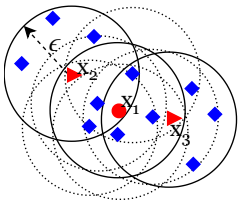


Fig. 2: Key idea of DBSVEC ($MinPts$=8)

Definition 5 (Sub-cluster): A sub-cluster $\mathcal{S}$ with respect to ϵ and $MinPts$ is a nonempty subset of $\mathbf{X}$ that satisfies: $\forall\mathbf{x}_i,\mathbf{x}_j\in\mathcal{S}$, there is a point $\mathbf{x}_k\in\mathcal{S}$ such that both $\mathbf{x}_i$ and $\mathbf{x}_j$ are density-reachable from $\mathbf{x}_k$ with respect to ϵ and $MinPts$.

For example, in Figure 2, the ϵ-neighborhood of $\mathbf{x}_1$ is a sub-cluster since any point is density-reachable from $\mathbf{x}_1$. This is formulated with the following lemma.

Lemma 1: $\forall\mathbf{x}_i\in\mathcal{S}$: if $|\mathcal{N}_\epsilon(\mathbf{x}_i)|\ge MinPts$, all points in the sub-cluster $\mathcal{S}$ are density-reachable from $\mathbf{x}_i$.

Proof: Consider an arbitrary core point $\mathbf{x}_i\in\mathcal{S}$ and another point $\mathbf{x}_j\in\mathcal{S}$. According to Definition 5, both $\mathbf{x}_i$ and $\mathbf{x}_j$ are density-reachable from a point $\mathbf{x}_k$. Since $\mathbf{x}_i$ is a core point, we conclude that $\mathbf{x}_k$ is also density-reachable from $\mathbf{x}_i$. Therefore, $\mathbf{x}_j$ is density-reachable from $\mathbf{x}_i$. □

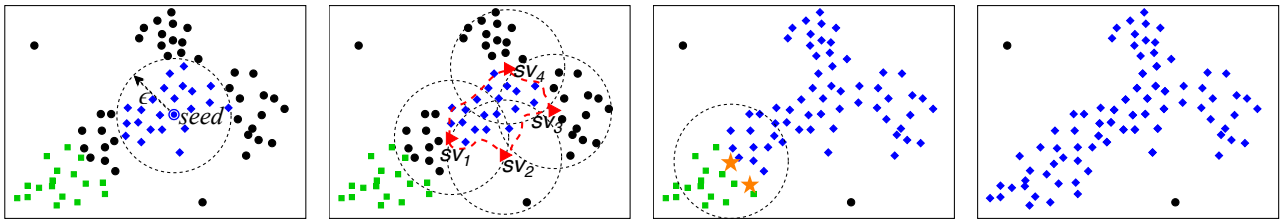
The DBSCAN algorithm can be seen as a process of expanding sub-clusters to form clusters until no more sub-clusters can be found or expanded.

When a sub-cluster such as the ϵ-neighborhood of $\mathbf{x}_1$ in Figure 2 is expanded, we find a subset of points in the sub-cluster whose range queries can cover all the points to be added to the sub-cluster. Such a subset should include those points near the boundary of the current sub-cluster (e.g., $\mathbf{x}_2$ and $\mathbf{x}_3$ in Figure 2). To identify such points, we propose to exploit *Support Vector Domain Descriptions* (SVDD) [14], a highly efficient kernel method for identifying boundary points. SVDD constructs a *rough* boundary (i.e., a hypersphere) that encloses the set of points using only a subset of the points near the boundary, i.e., the *support vectors*. So, we perform SVDD on a sub-cluster, and the obtained support vectors are used to expand the sub-cluster. As will be shown in Section III-C, performing range queries only on the support vectors can produce almost the same clusters as those produced from range queries on all the points. Therefore, the result of DBSVEC is usually very close to that of DBSCAN.

B. The DBSVEC Algorithm

DBSVEC has four major steps: initialization, support vector expansion, sub-cluster merging, and noise processing. Next, we use a running example as shown in Figure 3 to illustrate these steps. The pseudo-code is presented in Algorithm 2.

Initialization. According to Algorithm 2, DBSVEC scans the dataset and finds an unvisited point to activate a new sub-cluster (lines 1 to 3). When a point is visited, if it is not a core point (line 13), it is added into a list named *NoiseList* (line 14) that stores potential noise (line 15), and we proceed to the next unvisited point. If a point visited is a core point (line 4), it is used as the *seed* of a new sub-cluster (line 5). Such a point is denoted as the blue double circle in Figure 3a.



(a) Seed range query (b) SVDD based expansion (c) Merging sub-clusters (d) Final cluster

Fig. 3: DBSVEC running example (▶ represents a support vector, ★ represents an overlapping point, $MinPts$=15)

Algorithm 2 DBSVEC

Input: A finite set of points $\mathbf{X}=\{\mathbf{x}_1,...,\mathbf{x}_n\}^t$ in d-dimensional space $\mathbb{R}^d$ with $\mathbf{x}_i=(x_{i1},...,x_{id})^t$ being the ith point; a radius ϵ; a density threshold $MinPts$.
Output: Cluster ID of each point.

1: $Cid \leftarrow 0$
2: **for** each unclassified point $\mathbf{x}_i \in \mathbf{X}$ **do**
3: $\mathcal{N}_\epsilon(\mathbf{x}_i) \leftarrow$RangQuery($\mathbf{X}$,$\mathbf{x}_i$,$\epsilon$)
4: **if** $|\mathcal{N}_\epsilon(\mathbf{x}_i)| \geq MinPts$ **then**
5: $Cid \leftarrow Cid+1$, initialize $newClu$
6: **for** each point $\mathbf{x}_j \in N_\epsilon(\mathbf{x}_i)$ **do**
7: **if** $\mathbf{x}_j.id = unclassified$ or $noise$ **then**
8: $\mathbf{x}_j.id \leftarrow Cid$, add $\mathbf{x}_j$ into $newClu$
9: **else if** $\mathbf{x}_j.id \neq Cid$ **then**
10: **if** |RangQuery($\mathbf{X}$,$\mathbf{x}_j$,ϵ)|$\geq MinPts$ **then**
11: `Merge`($\mathbf{X}$,$\mathbf{x}_j.id$,Cid)
12: `svExpandCluster`($\mathbf{X}$,$newClu$,ϵ,$MinPts$,Cid)
13: **else**
14: $\mathbf{x}_i.id \leftarrow noise$
15: Add $\mathbf{x}_i$ into NoiseList L
16: `NoiseVerification`(L,ϵ,$MinPts$)

All the points within the ϵ-neighborhood of a seed $\mathbf{x}_i$ must be in the same cluster as $\mathbf{x}_i$, as formalized in the following corollary.

Corollary 1: $\forall \mathbf{x}_i \in \mathbf{X}$: if $|\mathcal{N}_\epsilon(\mathbf{x}_i)| \geq MinPts$, all points in $\mathcal{N}_\epsilon(\mathbf{x}_i)$ belong to the same cluster as $\mathbf{x}_i$.

Proof: Straightforward from Definition 5 and Lemma 1. □

Thus, we use the points in the ϵ-neighborhood of the seed as an initialized sub-cluster $\mathcal{S}$ (line 8), as shown by the blue diamonds surrounded by the dotted circle in Figure 3a.

Support vector expansion. We compute the support vectors for an initialized sub-cluster using SVDD (line 12 of Algorithm 2). We only keep the *core support vectors*, i.e., support vectors whose ϵ-neighborhood are dense (lines 1 to 6 of Algorithm 3).

Definition 6 (Core support vector): A point $\mathbf{x}_i \in \mathcal{S}_j$ is a core support vector if:

- in the SVDD model on $\mathcal{S}_j$, the Lagrange multiplier corresponding to $\mathbf{x}_i$, $\alpha_i > 0$; and
- $\mathcal{N}_\epsilon(\mathbf{x}_i)$ covers at least $MinPts$ points

Lemma 2: Given a sub-cluster $\mathcal{S}$, if there is a core support vector $\mathbf{x}_i \in \mathcal{S}$, then all the points in $\mathcal{N}_\epsilon(\mathbf{x}_i)$ and $\mathcal{S}$ belong to the same cluster.

Proof: According to Lemma 1, points in $\mathcal{N}_\epsilon(\mathbf{x}_i)$ and $\mathcal{S}$ are density-reachable from $\mathbf{x}_i$. By connectivity of Definition 4, $\mathcal{N}_\epsilon(\mathbf{x}_i)$ and $\mathcal{S}$ belong to the same cluster. □

Algorithm 3 `svExpandCluster`

Input: A finite set of points $\mathbf{X}=\{\mathbf{x}_1,...,\mathbf{x}_n\}^t$ in d-dimensional space $\mathbb{R}^d$ with $\mathbf{x}_i=(x_{i1},...,x_{id})^t$ being the ith point; a finite set of points $sub\text{-}cluster$ whose points from $\mathbf{X}$; a radius ϵ; a density threshold $MinPts$; current ID Cid.
Output: The expanded sub-cluster.

1: $Model \leftarrow$`SVDD`(sub-cluster)
2: $sv \leftarrow Model$.SupportVectorSet
3: lastSize $\leftarrow$|sub-cluster|
4: **for** $i \leftarrow 1$ **to** $|sv|$ **do**
5: $\mathcal{N}_\epsilon(sv[i]) \leftarrow$RangeQuery($\mathbf{X}$,sv[$i$],$\epsilon$)
6: **if** $|\mathcal{N}_\epsilon(sv[i])| \geq MinPts$ **then**
7: **for** $j \leftarrow 1$ **to** $|\mathcal{N}_\epsilon(sv[i])|$ **do**
8: **if** $\mathcal{N}_\epsilon(sv[i])[j].id = unclassified$ or $noise$ **then**
9: $\mathcal{N}_\epsilon(sv[i])[j].id \leftarrow Cid$
10: Add $\mathcal{N}_\epsilon(sv[i])[j]$ into sub-cluster
11: **else if** $\mathcal{N}_\epsilon(sv[i])[j].id \neq Cid$ **then**
12: **if** |RangQuery($\mathbf{X}$,$\mathcal{N}_\epsilon(sv[i])[j]$,$\epsilon$)|$\geq MinPts$ **then**
13: `Merge`($\mathbf{X}$,$\mathcal{N}_\epsilon(sv[i])[j].id$,$Cid$)
14: **if** $|sub\text{-}cluster| >$ lastSize **then**
15: `svExpandCluster`($\mathbf{X}$,sub-cluster,ϵ,$MinPts$,Cid)

We repeatedly compute the core support vectors over the (continuously) expanding sub-cluster using SVDD and add the points in the ϵ-neighborhood of the core support vectors to the sub-cluster until no new core support vectors can be found (lines 7 to 10 of Algorithm 3). If all support vectors are non-core points or the sub-cluster cannot be expanded further, we go back to *initialization* to look for a new sub-cluster. In Figure 3b, the red triangles represent the support vectors found, and the red dashed line (a by-product of SVDD) is the boundary formed by the high-dimensional sphere mapping back to the original space. We see that SV_1, SV_3 and SV_4 are core support vectors, while SV_2 is a non-core support vector as the number of its surrounding neighbors is less than $MinPts$. The sub-cluster is expanded from SV_1, SV_3 and SV_4. Algorithm 3 summarizes support vector expansion.

Sub-cluster merging (Line 11 of Algorithm 2 and line 13 of Algorithm 3). During *initialization* and *support vector expansion*, a point to be added to the expanding sub-cluster may have been assigned to another existing sub-cluster, we call such a point an *overlapping point*. If an overlapping point is also a core point, the existing sub-cluster should be merged with the expanding sub-cluster by Lemma 3.

Lemma 3: Given two sub-clusters $\mathcal{S}_i$ and $\mathcal{S}_j$, if there is a core support vector $\mathbf{x} \in \mathcal{S}_i \wedge \mathbf{x} \in \mathcal{S}_j$, then all the points in $\mathcal{S}_i$ and $\mathcal{S}_j$ belong to the same cluster.

Proof: Similar to Lemma 2. □

Figure 3 shows an existing sub-cluster denoted by green squares. In Figure 3c, overlapping points are found (the orange star) during *support vector expansion*. After confirming that an overlapping point is a core point, the blue and the green sub-clusters are merged (Figure 3d).

Noise verification (Line 16 of Algo. 2). After all the points are visited (assigned a cluster label or stored as a potential noise point in $NoiseList$), we check whether there are core points in the ϵ-neighborhood $\mathcal{N}_\epsilon(NoiseList[i])$ of each potential noise point $NoiseList[i]$. If there is no core point, $NoiseList[i]$ is confirmed as a noise point. Otherwise, $NoiseList[i]$ is assigned to the cluster of its nearest core point. Note that $\mathcal{N}_\epsilon(NoiseList[i])$ has been obtained in *initialization*.

C. Accuracy Analysis

An approximate DBSCAN algorithm is more "accurate" if its clustering result is more similar to that of DBSCAN [5], [11], [22]. Specifically, we follow Lulli et al. [22] and use *recall* to measure the accuracy of clustering results. This recall metric is computed as the ratio of point pairs that share the same cluster in the clustering results of both DBSCAN and an approximate DBSCAN algorithm to be evaluated. A larger recall means a higher accuracy. Next, we show that DBSVEC produces highly accurate results, with the help of the following symbols.

- $\mathscr{C}_D$ denotes the set of clusters produced by DBSCAN with parameters (ϵ, *MinPts*).
- $\mathscr{C}_S$ denotes the set of clusters produced by DBSVEC with parameters (ϵ, *MinPts*).

We first show that any cluster produced by DBSVEC must be a subset of some cluster produced by DBSCAN.

Theorem 1 (Necessity Guarantee): Given dataset $\mathbf{X}$ and parameters (ϵ, *MinPts*), for any cluster $Cl_S \in \mathscr{C}_S$, there is a cluster $Cl_D \in \mathscr{C}_D$ such that $Cl_S \subseteq Cl_D$.

Proof: Consider an arbitrary core point $\mathbf{x}_i \in Cl_S$. Point $\mathbf{x}_i$ must also be a core point in DBSCAN. Based on Lemma 1, all points in Cl_S are density-reachable from $\mathbf{x}_i$. According to the maximality condition in Definition 4, if $Cl_D \in \mathscr{C}_D$ contains $\mathbf{x}_i$, then Cl_D contains all the points density-reachable from $\mathbf{x}_i$. Hence, all the points in Cl_S must also be in Cl_D. □

Meanwhile, the noise points and border points identified by DBSVEC are also the same as those identified by DBSCAN.

Theorem 2 (Border Point Guarantee): Given dataset $\mathbf{X}$ and parameters (ϵ, *MinPts*), the border points in any cluster $Cl_S \in \mathscr{C}_S$ are the same as the border points in some cluster $Cl_D \in \mathscr{C}_D$, if Cl_S and Cl_D have the same core points.

Proof: Let $\mathbf{x}_i$ be an arbitrary border point in Cl_D. According to the connectivity condition in Definitions 4, there exists a core point $\mathbf{x}_j \in Cl_D$ from which $\mathbf{x}_i$ is density-reachable. In DBSVEC, $\mathbf{x}_i$ is assigned to the same cluster as $\mathbf{x}_j$ by *support vector expansion* or *noise verification*. On the other side, consider an arbitrary border point $\mathbf{x}_i \in Cl_S$. According to the maximality condition in Definitions 4, there exists $\mathbf{x}_i \in Cl_D$. Hence, all border points of Cl_D are the same as those of Cl_S. □

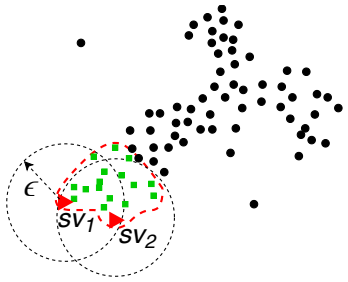


Fig. 4: A case of a sub-cluster stopping expansion

Theorem 3 (Noise Point Guarantee): Given dataset $\mathbf{X}$ and parameters (ϵ, *MinPts*), the noises found by DBSVEC and DBSCAN are the same.

Proof: Let Cl_{noise} be the set of noise points in DBSCAN. Since *NoiseList* in DBSVEC is the set of potential noise, it is easy to know $Cl_{noise} \subseteq$ *NoiseList*. In the final step of DBSVEC, *noise verification* confirms the noises in *NoiseList* and satisfies Cl_{noise}=*NoiseList*. □

It is interesting to examine whether a cluster in DBSCAN is also a subset of some cluster in DBSVEC. Unfortunately, this does not always hold, because using only the support vectors to expand the sub-clusters does not guarantee the maximality of the clusters in DBSVEC.

The implication is that DBSVEC may divide a cluster of DBSCAN into multiple clusters, but it will not put multiple clusters of DBSCAN into a single cluster.

Next, we give the conditions under which DBSVEC may divide a cluster of DBSCAN into multiple clusters. We use the following symbols to facilitate the discussion.

- Cl_D denotes a cluster of DBSCAN with parameters (ϵ, *MinPts*).
- $\mathscr{S}$ denotes a sub-cluster of DBSVEC with parameters (ϵ, *MinPts*).
- $\mathscr{S} \subset Cl_D$, and $\mathscr{S}$ does not expand to become Cl_D.

Condition 1: In the step of *support vectors expansion*: the sub-cluster $\mathscr{S}$ stops expanding before all the core points in the cluster Cl_D are found.

- There is a core point in Cl_D not yet assigned to $\mathscr{S}$.
- The support vectors obtained by computing SVDD on $\mathscr{S}$ are non-core points, or the ϵ-neighborhood of the core support vectors do not contain new points not yet in $\mathscr{S}$.

Figure 4 illustrates a case where a sub-cluster stops expanding. The sub-cluster (denoted by squares and triangles) is located at the bottom-left part of the cluster, and its support vectors (denotes by triangles) happen to also locate at the bottom-left part of the cluster. Thus, expanding from these support vectors does not grow the sub-cluster towards the full cluster. Note that this case does not necessarily result in incorrect clustering results because the other points may form a sub-cluster which expands to merge with this sub-cluster.

Condition 2: In the step of *sub-cluster merging*: None of the core points in $\mathscr{S}$ is found in the *initialization* and *support vector expansion* steps of any other sub-clusters (i.e., $\mathscr{S}$ has never been merged).

- The ϵ-neighborhood of all the new seeds do not contain any core points of $\mathscr{S}$.
- The ϵ-neighborhood of all the core support vectors obtained by computing SVDD on the other sub-clusters do not contain any core points of $\mathscr{S}$.

The above conditions are rarely met at the same time, which is confirmed by experiments in Section V-B on datasets of ten different distributions. Therefore, the clustering result of DBSVEC is very close and often identical to that of DBSCAN.

D. Complexity Analysis

Given a dataset $\mathbf{X}$ with n points in a d-dimensional space and two parameters $\epsilon \in \mathbb{R}^+$ and $MinPts \in \mathbb{N}^+$, let s be the number of cluster seeds, l be the size of *NoiseList*, $O(\widetilde{n})$ be the average set size for SVDD computation, m be the number of sub-cluster mergers, and k be the total number of support vectors. We analyze the running time of DBSVEC as follows.

Initialization requires at most $O(s)$ range queries to seed the sub-clusters. In *support vector expansion*, training an improved SVDD model using the technique to be detailed in Section IV needs $O(\widetilde{n})$ time. When $\widetilde{n}$ is larger, the expected number of times of SVDD training $O(n/\widetilde{n})$ is smaller. Hence, all SVDD training together take $O(n)$ time (see Section IV-D). Querying whether the support vectors are core points requires $O(kn)$ time. The number of support vectors ranges from 1 to $\widetilde{n}$ (worst-case) in each SVDD training. Based on an optimization technique in Sections IV-B and IV-C, the total number of support vectors k is much smaller than the size of the dataset n, which can be controlled by the parameters penalty factors ν and kernel width σ. When sub-clusters find overlapping points, *sub-cluster merging* needs to perform range queries on the overlapping points, which take $O(mn)$ time. The last step *noise verification* consumes less than $O(MinPts \cdot ln)$ time to identify true noise from potential noise, where l depends on the number of noise in the dataset. Overall, DBSVEC requires $O((s+1+k+m+MinPts \cdot l)n) = O(\theta n)$ time. As analyzed above, s, k, m, l are all far smaller than n, i.e., $\theta \ll n$. This has also been validated by extensive experiments (see Section V-C). Therefore, DBSVEC runs much faster than DBSCAN which has a time complexity of $O(n^2)$. Note that the $O(n)$ factor in our cost is for performing range queries. Using spatial indices can further bring down this factor [23].

DBSVEC needs $O(n+\widetilde{n}+MinPts \cdot l)$ space for storing the cluster labels, the target data for SVDD computation, and the *NoiseList*. This cost is linear to the dataset size n. While DBSCAN [1] and existing approximate techniques [5], [11] also have a linear space cost, they need to store and maintain an extra index which is not needed by DBSVEC.

IV. Improving SVDD for DBSVEC

In DBSVEC, *support vector expansion* is a repeated step with non-trivial costs. The focus is on how to further optimize this procedure towards higher clustering accuracy and efficiency. We first reformulate SVDD in Section IV-A and introduce a penalty weight for each point to guide the support vector computation process towards selecting points on the cluster boundary. This helps improve the clustering accuracy. We further propose an incremental learning technique and a kernel parameter value selection strategy in Section IV-B, which help improve the clustering efficiency. We discuss the trade-off between accuracy and efficiency in Section IV-C and the costs of proposed techniques in Section IV-D.

A. Improving Accuracy

In the SVDD objective function (Eq. 1), the penalty factor C is a trade-off parameter controlling how much the slack variables ξ_i are penalized, while ξ_i is used to represent how far away a point can fall outside of the sphere constructed by SVDD (the support vectors are located either on or outside the sphere and hence have larger slack variable values) [24]. In the original SVDD model, the same penalty factor C is used for every data point without discrimination [14]. However, this is not applicable for DBSVEC since in the clustering process, points newly added to a sub-cluster or far from the center of a sub-cluster should have smaller penalty factors, to allow larger slack variable values and hence a higher probability for such points to be selected as support vectors. The rationale is that newly expanded data points and those far from the center of sub-cluster in kernel space are more likely to locate either on or outside the sub-cluster sphere. We thus should encourage such points to be used as support vectors.

Based on the observation above, we assign each data point an individual penalty factor indicating its possibility of being a support vector. We define the penalty weight of point $\mathbf{x}_i$ to be exponential to the number of times that $\mathbf{x}_i$ has participated in support vector computation and inversely proportional to the distance between $\mathbf{x}_i$ and the center of the sub-cluster in the kernel space. We first introduce *kernel distance function* and *memory factors* as follows.

The kernel distance function is defined as:

$$\mathcal{D}(\mathbf{x}) = \left\| \Phi(\mathbf{x}) - \frac{1}{\widetilde{n}} \sum_{i=1}^{\widetilde{n}} \Phi(\mathbf{x}_i) \right\|_{\mathcal{H}}^2 = \frac{1}{\widetilde{n}^2} \sum_{i,j=1}^{\widetilde{n}} K(\mathbf{x}_i,\mathbf{x}_j) + K(\mathbf{x},\mathbf{x}) - \frac{2}{\widetilde{n}} \sum_{i=1}^{\widetilde{n}} K(\mathbf{x}_i,\mathbf{x}) \tag{5}$$

where $\widetilde{n}$ is the number of target data points, in this case, the size of the currently expanding sub-cluster; K is the kernel function where $K_{ij} = K(\mathbf{x}_i,\mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$; Φ is a nonlinear mapping of the input space into a Hilbert space $\mathcal{H}$. We use the *Gaussian kernel*:

$$K(\mathbf{x}_i,\mathbf{x}_j) = \exp\left(-\frac{\left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2}{2\sigma^2} \right) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j), \qquad \sigma > 0 \tag{6}$$

where σ is the root mean square (RMS) width parameter of the kernel function, and we will discuss how to decide its value in the following subsections. For a given cluster, according to Eq. 6, $K(\mathbf{x},\mathbf{x}) \equiv 1$ and $\frac{1}{\widetilde{n}^2} \sum_{i,j=1}^{\widetilde{n}} K_{i,j}$ are constants in Eq. 5.

The memory factor λ is a coefficient greater than 1, which is used to define the penalty weight:

$$\omega_i = \lambda^{t_i} \left(1 - \frac{\mathcal{D}(\mathbf{x}_i)}{\max\limits_{j=1,\ldots,\widetilde{n}} \mathcal{D}(\mathbf{x}_j)} \right) \qquad \forall i = 1,2,\ldots,\widetilde{n} \tag{7}$$

where t_i is the number of times that $\mathbf{x}_i$ participates in SVDD training. Since λ^{t_i} increases exponentially with t_i, old points are generally given larger penalty, while points newly added to

the target dataset have smaller penalty. As the penalty weight ω_i is inversely proportional to the kernel distance $\mathcal{D}(\mathbf{x})$, data points far from the target data center can get large slack variable values and are more likely to become support vectors.

Using penalty weights, the optimization problem becomes:

$$\min_{R\in\mathbb{R},\mathbf{a}\in\mathcal{H}} f(R,\mathbf{a},\xi_i)=R^2+C\sum_{i=1}^{\widetilde{n}}\omega_i\xi_i$$
$$s.t. \quad \|\Phi(\mathbf{x}_i)-\mathbf{a}\|^2_{\mathcal{H}}\le R^2+\xi_i, \quad \xi_i\ge 0 \quad \forall i \tag{8}$$

Here we use a nonlinear function $\Phi(\mathbf{x})$ (whose inner product is Gaussian kernel Eq. 6) to obtain a sphere that can bound the data points tighter [14]. By adding Lagrangian multipliers $\alpha_i,\beta_i\ge 0$ for the constraints of Eq. 8, we have

$$\begin{aligned}\mathcal{L}(R,\mathbf{a},\xi)=&-\sum_{i=1}^{\widetilde{n}}\alpha_i(R^2+\xi_i-\|\Phi(\mathbf{x}_i)-\mathbf{a}\|^2_{\mathcal{H}})\\ &-\sum_{i=1}^{\widetilde{n}}\beta_i\xi_i+R^2+C\sum_{i=1}^{\widetilde{n}}\omega_i\xi_i \qquad \alpha_i\ge 0,\ \beta_i\ge 0\end{aligned} \tag{9}$$

Letting the derivative of $\mathcal{L}$ to zero with respect to R, $\mathbf{a}$, ξ_i, respectively, leads to

$$\sum_{i=1}^{\widetilde{n}}\alpha_i=1, \quad \mathbf{a}=\sum_{i=1}^{\widetilde{n}}\alpha_i\Phi(\mathbf{x}_i), \quad \alpha_i=\omega_i C-\beta_i \tag{10}$$

Substituting Eq. 10 back into Eq. 9, the dual optimal problem of Eq. 8 can be rewritten as

$$\begin{aligned}\max_{\alpha_i} \quad \mathcal{L}_D=&\sum_{i=1}^{\widetilde{n}}\alpha_i K(\mathbf{x}_i,\mathbf{x}_i)-\sum_{i=1}^{\widetilde{n}}\sum_{j=1}^{\widetilde{n}}\alpha_i\alpha_j K(\mathbf{x}_i,\mathbf{x}_j)\\ s.t. \quad &0\le\alpha_i\le\omega_i C, \qquad \sum_{i=1}^{\widetilde{n}}\alpha_i=1\end{aligned} \tag{11}$$

Here, the difference in the dual formula between Eq. 4 and Eq. 11 are the upper bounds of the Lagrange multipliers α_i and the use of the kernel function. The upper bounds in Eq. 11 are no longer the same. Instead, they are controlled by the corresponding penalty weights. Note that, point $\mathbf{x}_i$ with $0<\alpha_i\le C$ is a support vector on the boundary around the target data. Whether a point is within the sphere can be determined by the following discrimination function:

$$\begin{aligned}\mathcal{F}(\mathbf{x})=(\Phi(\mathbf{x})-\mathbf{a})(\Phi(\mathbf{x})-\mathbf{a})^T=&K(\mathbf{x},\mathbf{x})-2\sum_{i=1}^{\widetilde{n}}\alpha_i K(\mathbf{x}_i,\mathbf{x})\\ &+\sum_{i=1}^{\widetilde{n}}\sum_{j=1}^{\widetilde{n}}\alpha_i\alpha_j K(\mathbf{x}_i,\mathbf{x}_j)\ \le\ R^2\end{aligned} \tag{12}$$

When the distance between a point $\mathbf{x}_i$ and the sphere center $\mathbf{a}$ is smaller than the radius, $\mathbf{x}_i$ is within the sphere.

B. Improving Efficiency

Next, we consider improving the time efficiency of SVDD.

1) Incremental Learning: During *support vector expansion*, a growing number of data points are involved in SVDD training until all the data points in the full cluster are identified. The data points repeatedly used for computing support vectors contribute little to the SVDD model but take a significant portion of the computation. To improve the efficiency, we propose an incremental learning method with a focus on data points newly added to the target dataset.

We use a learning threshold $\mathcal{T}$ to control the number of times that a point can be used in the target dataset for SVDD computation. Once a point is added to the target dataset, it is assigned a counter t_i initialized to 0. After SVDD training is done on the current target dataset, the t_i value of each target data point is increased by 1, and data points with $t_i>\mathcal{T}$ are eliminated from the target dataset for SVDD training (over the expanded sub-cluster). By doing so, the algorithm can learn support vectors from the newly expanded data and discover more points to be added into the sub-cluster, rather than re-discovering the same support vectors used before.

Intuitively, when $\mathcal{T}$ is large, more "old" points are retained, resulting in higher SVDD training time. On the other hand, if $\mathcal{T}$ is set to 0, it is equivalent to computing SVDD using only data points newly added to the sub-cluster. Experimental results show that, when the threshold $\mathcal{T}$ is in the range of 2 to 4, our incremental learning method can improve algorithm efficiency with negligible impact on accuracy. As a result, we use $\mathcal{T}=3$ in our experiments in Section V.

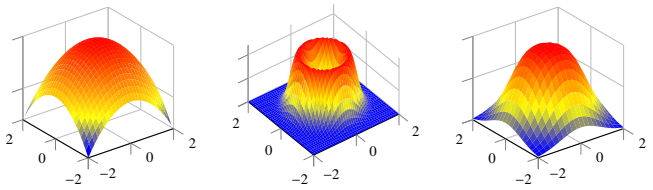
2) Kernel Parameter Value Selection: To find the optimal boundary description of a sub-cluster, SVDD uses the Gaussian kernel to project data into a high dimensional space by nonlinear transformation. The kernel parameter σ determines the degree of nonlinear transformation. Using the Taylor series, we can expand the Gaussian kernel into *infinite* dimensions to observe the effects of σ.

$$\begin{aligned}K(\mathbf{x}_i,\mathbf{x}_j)=&\exp\left(-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2\sigma^2}\right)\\ =&\exp\left(-\frac{\mathbf{x}_i^2+\mathbf{x}_j^2}{2\sigma^2}\right)\left(1+\frac{1}{1!}\left(\frac{\mathbf{x}_i\mathbf{x}_j}{\sigma^2}\right)+\frac{1}{2!}\left(\frac{\mathbf{x}_i\mathbf{x}_j}{\sigma^2}\right)^2+\frac{1}{3!}\left(\frac{\mathbf{x}_i\mathbf{x}_j}{\sigma^2}\right)^3+\cdots\right)\end{aligned} \tag{13}$$

When σ is smaller, weights $1/(n!\sigma^n)$ on high-dimensional features decay slowly and hence there is a higher degree of nonlinear transformation. A higher degree of nonlinearity leads to a tighter boundary of the target dataset formed by SVDD, which can better reflect the shape of data. However, a higher degree of nonlinearity does not necessarily lead to better support vectors. This is because, under a higher degree of nonlinearity, SVDD may produce support vectors not at the boundary of the target dataset (i.e., overfitting [25]). This may reduce the efficiency of DBSVEC. Thus, our kernel parameter selection strategy aims to find a lower bound of σ that helps form the optimal boundary description of a sub-cluster while it is not too high to trigger the overfitting.

We focus on an extreme scenario of data distribution where the interior of the dataset is empty [26]. With the same kernel parameter settings, since there are no data points in the interior, SVDD tends to regard the interior sparse space as the outer space of the hypersphere in the kernel space. This will cause overfitting [14], [25]. Next, we show how to derive the lower bound value of the kernel parameter to avoid this false perception on SVDD.

Without loss of generality, we consider a two-dimensional data space, and it is straightforward to generalize to high-



(a) Original space (b) Kernel Space $\sigma=0.5r$ (c) Kernel Space $\sigma=r/\sqrt{2}$

Fig. 5: Distance measures in the original and kernel spaces

dimensional spaces. In the above scenario, the data is distributed on a sphere given by the following equation.

$$\mathbf{s}_i=\begin{pmatrix} r\cos(\frac{2\pi}{n}(i-1)) \\ r\sin(\frac{2\pi}{n}(i-1)) \end{pmatrix}, \quad where \quad i=1,...,n \tag{14}$$

Note that according to Mercer's Theorem [27], in the solution of SVDD, the kernel function is the sum of inner products (cf. Eq. 11 and Eq. 12). In discrimination functions $\mathcal{F}(\mathbf{x})$ (Eq. 12), both $K(\mathbf{x},\mathbf{x})$ and $\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j K(\mathbf{x}_i,\mathbf{x}_j)$ are constants whereas $\sum_{i=1}^{n}\alpha_i K(\mathbf{x}_i,\mathbf{x})$ determines the distance between point $\mathbf{x}$ and the sphere center $\mathbf{a}$. As the space is symmetric and $\sum_{i=1}^{n}\alpha_i=1$, we can choose $\alpha_i=\frac{1}{n}$. The *solution function* is therefore:

$$f(\mathbf{x})=\frac{1}{n}\sum_{i=1}^{n}K(\mathbf{x},\mathbf{s}_i)=\frac{1}{n}\sum_{i=1}^{n}\Phi(\mathbf{x})\Phi(\mathbf{s}_i)=\frac{1}{n}\sum_{i=1}^{n}\exp\left(-\frac{\|\mathbf{x}-\mathbf{s}_i\|^2}{2\sigma^2}\right) \tag{15}$$

where $f(\mathbf{x})$ represents the opposite of the distance between point $\mathbf{x}$ and the sphere center in kernel space, and the greater the value of $f(\mathbf{x})$, the closer $\mathbf{x}$ is to the sphere center. When n goes to infinity, the following solution function:

$$\lim_{n\to\infty} f(\mathbf{x})=\frac{1}{2\pi}\int_0^{2\pi}\exp\left(-\frac{1}{2}\left(\frac{r}{\sigma}\right)^2\left\|\frac{\mathbf{x}}{r}-\begin{pmatrix}\cos\omega\\ \sin\omega\end{pmatrix}\right\|^2\right)d\omega \tag{16}$$

only depends on the circle radius r and the kernel parameter σ. Figures 5a and 5b show the plots of $f(\mathbf{x})$ in the original space and a kernel space (with a smaller $\sigma=0.5r$). The function in the original space forms a "unimodal" shape with a peak at the origin while in the kernel space, it forms a "crater" shape with a basin in the center. This means that in the kernel space, the point closer to the sphere center in the original space may be regarded as farther away from the sphere center in the kernel space, i.e., the distance measures are inconsistent. Consequently, this may lead to the selection of internal points (rather than boundary points) as support vectors, which may impact the clustering efficiency.

To obtain appropriate kernel parameter values, we compute the gradient and second-order partial derivatives of f:

$$\frac{\partial f}{\partial x_1}=\frac{1}{n\sigma^2}\sum_{i=0}^{n-1}\left(r\cos\left(\frac{2\pi i}{n}\right)-x_1\right)\exp\left(-\frac{\|\mathbf{x}-\mathbf{s}_i\|^2}{2\sigma^2}\right) \tag{17}$$

$$\frac{\partial^2 f}{\partial x_1^2}=\frac{1}{n\sigma^2}\sum_{i=0}^{n-1}\left(\frac{\left(r\cos\left(\frac{2\pi i}{n}\right)-x_1\right)^2}{\sigma^2}-1\right)\exp\left(-\frac{\|\mathbf{x}-\mathbf{s}_i\|^2}{2\sigma^2}\right) \tag{18}$$

It is known from the necessary and sufficient conditions of the extreme value that the extreme value of f is obtained at the origin. Using the symmetry of the space or letting n be infinite, we obtain:

$$\begin{aligned}\lim_{n\to\infty}\left.\frac{\partial^2 f}{\partial x_1^2}\right|_{\mathbf{x}=(0,0)} &=\frac{1}{2\pi\sigma^2}\exp\left(-\frac{r^2}{2\sigma^2}\right)\int_0^{2\pi}\left(-1+\left(\frac{r}{\sigma}\right)^2\cos^2\omega\right)d\omega \\ &=\frac{1}{\sigma^2}\exp\left(-\frac{r^2}{2\sigma^2}\right)\left(-1+\left(\frac{r}{\sqrt{2}\sigma}\right)^2\right)\end{aligned} \tag{19}$$

where for $\sigma>\frac{r}{\sqrt{2}}$, the Gaussian kernel function f takes the global maximum value at the origin, and hence avoids overfitting. For $\sigma<\frac{r}{\sqrt{2}}$, function f has a local minimum value at the origin and a basin shape near the origin. Figure 5c shows the critical case with $\sigma=\frac{r}{\sqrt{2}}$.

The lower bound ($\frac{r}{\sqrt{2}}$) of the kernel parameter can ensure that, for data with radius r, using Gaussian kernel functions can avoid the overfitting of the SVDD model. In our experiments, we use $\sigma=\frac{r}{\sqrt{2}}$, where r is the distance from the center of the target data to the point farthest from it.

C. Balancing Accuracy and Efficiency

We now consider the penalty factor C, which controls the trade-off between the volume of the sphere and the accuracy of data description in SVDD. *One-Class Support Vector Machine* (OC-SVM) [28] replaces C with a new parameter ν. Parameter ν is a reparametrization of C and therefore they are mathematically equivalent[1] ($C=1/\nu\widetilde{n}$). Schölkopf and Smola [29] show that $\nu\in(0,1)$ is an upper bound on the fraction of boundary support vectors (BSVs) and a lower bound on the fraction of support vectors (SVs). Increasing ν and hence the number of support vectors will enhance the accuracy but reduce the efficiency of the algorithm, and vice versa.

As a rule of thumb, the number of support vectors should increase with the growth of sub-cluster and data dimensionality. We give an empirical choice of penalty factors:

$$\nu=\nu^*=\frac{d\sqrt{\log_{MinPts}\widetilde{n}}}{\widetilde{n}}, \qquad C=\frac{1}{\nu\widetilde{n}} \tag{20}$$

where d is the dimensionality. According to our experiments, this adaptive penalty factor can achieve a balance between accuracy and efficiency. It is worth pointing out that DBSVEC degenerates to DBSCAN when ν approaches 1.

D. Complexity Analysis

We analyze the costs of our improved SVDD. Let $\widetilde{n}$ be the size of target data in SVDD computation. Computing penalty weights and the kernel parameter values consume $O(\widetilde{n})$ time. Training SVDD needs to solve a quadratic programming (QP) problem. We exploit the *Sequential Minimal Optimization* (SMO) [30] approach to break the large QP problem into a series of the small QP problems, which results in a linear time and space complexity $O(\widetilde{n})$ to the target dataset size and dimensionality [28].

In the incremental learning technique, the size of the target set $\widetilde{n}$ is usually small with $c\cdot MinPts\leq\widetilde{n}\ll n$, where c is a constant positively correlated to the learning threshold $\mathcal{T}$

[1]Radial Basis Function kernel (specifically, a Gaussian RBF kernel) has the properties that $K(\mathbf{x}_i,\mathbf{x}_i)=1$ for all $\mathbf{x}_i\in\mathbf{X}$. In this case, with $C=1/(\nu\widetilde{n})$, the problems of SVDD and OC-SVM are identical, and both methods learn the same decisions functions.

TABLE III: Clustering Accuracy over Open Datasets

Method \ Dataset (n,d)	Seeds 210,7	Map-Jo. 6014,2	Map-Fi. 13467,2	Breast. 669,9	House 34112,3	Miss. 6480,16	Dim32 1024,32	Dim64 1024,64	Data31 3100,2	t4.8k 8000,2	t7.10k 10000,2
DBSVEC$_{min}$	1.000	1.000	1.000	0.976	1.000	1.000	1.000	1.000	1.000	1.000	0.997
DBSVEC	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
ρ-Appr	1.000	1.000	0.995	0.846	0.993	1.000	0.887	0.887	0.885	1.000	1.000
DBSCAN-LSH	0.847	0.832	1.000	0.997	0.889	0.831	0.994	1.000	1.000	0.793	0.645

and the neighborhood size $|\mathcal{N}_\epsilon(\mathbf{x}_i)|$. When the radius ϵ is large, $O(\widetilde{n})$ will be large, while the number of times to compute support vectors, $O(\frac{n}{\widetilde{n}})$, will be reduced. Overall, as our experiments in Section V-C show, the incremental learning technique helps reduce the running time by up to an order of magnitude.

V. Experiments

In this section, we present an empirical evaluation of the proposed algorithms. All the experiments are done on a machine with a 3.1GHz CPU and 16GB memory running macOS 10.13.2. The algorithms are implemented in C++ and compiled using Apple LLVM 9.0. Our SVDD implementation is developed based on *lib-svm* [2].

A. Baseline Algorithms

We compare DBSVEC with the following algorithms:

- **R-DBSCAN**: the original DBSCAN algorithm implementation [1] using an in-memory R-tree [7]. We use the clustering result of this algorithm as the ground truth for evaluating the clustering accuracy of DBSVEC.
- **kd-DBSCAN**: a DBSCAN implementation using an in-memory kd-tree [6]. This is a popular Python tool-kit[3] we have also considered following a previous study [31]
- **DBSCAN-LSH**: a hashing-based approximate DBSCAN algorithm [11] using p-stable hashing functions.
- **ρ-Approximate**: the state-of-the-art DBSCAN approximation algorithm [5] with a quadtree-like gird.
- **NQ-DBSCAN**: a recently proposed fast DBSCAN algorithm using local neighborhood searching technique [19] coded in MATLAB.
- **k-MEANS**: a popular partitioning-based clustering algorithm [32].

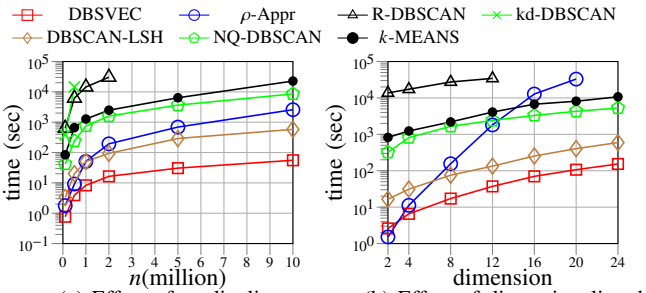
We set the upper limit of the running time to 10 hours. In ρ-Approximate[4], ρ=0.001 as recommended [5]. DBSCAN-LSH uses eight p-stable hashing functions [11]. In the efficiency experiments (Section V-C), following [5], if R-DBSCAN or kd-DBSCAN do not terminate in 10 hours or run out of memory, no results are reported (Figures 6 and 7).

B. Clustering Accuracy

2D visualization. To demonstrate the effectiveness of DBSVEC, we use a 2D dataset *t4.8k* (with cardinality 8000) which is a classic benchmark dataset for verifying clustering quality [13]. We use $MinPts$=20 and ϵ=8.5. From Figure 1 (Section I), we can see that DBSVEC and DBSCAN produce equally good clustering results on the dataset.

TABLE IV: Clustering validation.
"C" stands for compactness (Higher values are preferred), "S" for separation (Lower values are preferred).

Algorithm	Miss. (d=16)		Breast. (d=9)		Dim64 (d=64)	
	C	S	C	S	C	S
DBSVEC	**0.424**	**0.833**	**0.667**	**0.687**	**0.966**	**0.050**
k-MEANS	0.087	2.268	0.597	0.761	0.966	0.050



(a) Effect of cardinality n (b) Effect of dimensionality d

Fig. 6: Scalability tests

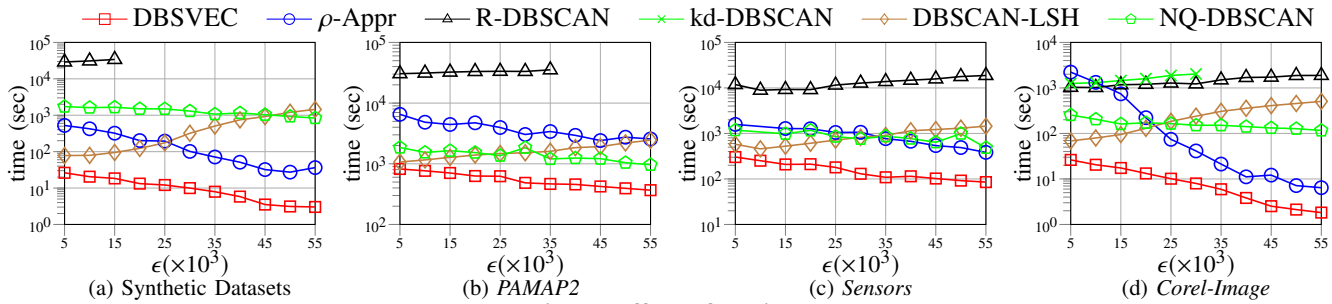
Statistical results. Next, we examine the accuracy of DBSVEC on other open datasets from different domains including *Seeds* [33], *Dim32* and *Dim64* [34], *Map-Joensuu* and *Map-Finland*[5], *D31* [35], *Breast-Cancer* [33], *Miss-America* [36], *House* [36], *t4.8k* and *t7.10k* [13].

As discussed in Section III-C, we follow Lulli et al. [22] and use *recall* to measure clustering accuracy. Note that running DBSCAN and computing the recall are expensive on larger datasets [22], thus we use these relatively small datasets.

Table III shows the clustering quality results of approximate algorithms, where DBSVEC and DBSVEC$_{min}$ represent the proposed algorithm running with the optimal value (ν^*) and minimum value (ν=$1/\widetilde{n}$) of ν as described in Section IV-C, respectively. We see that DBSVEC produces perfect recall for all of the datasets when ν=ν^*. Even when using the minimum ν value, the recall of DBSVEC is larger than or equal to those of ρ-Approximate and DBSCAN-LSH for all the datasets except t7.10k. The results confirm Lemmas 2 and 3 in Section III-C that DBSVEC produce clustering results very similar to those of DBSCAN.

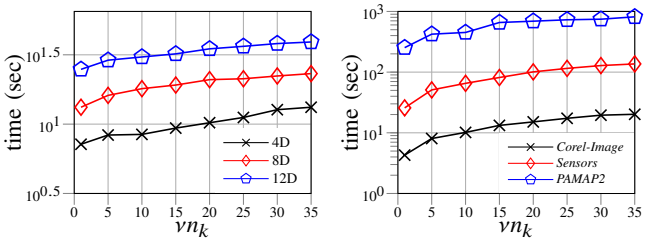
We also use internal validation metrics *Compactness* [37] and *Separation* [38] to compare the clustering quality of our algorithm with k-MEANS. From Table IV, we observe that DBSVEC consistently produces higher quality clustering results than k-MEANS.

[2]https://www.csie.ntu.edu.tw/~cjlin/libsvm/
[3]http://scikit-learn.org/stable
[4]https://sites.google.com/site/junhogan/
[5]http://cs.uef.fi/mopsi/data/



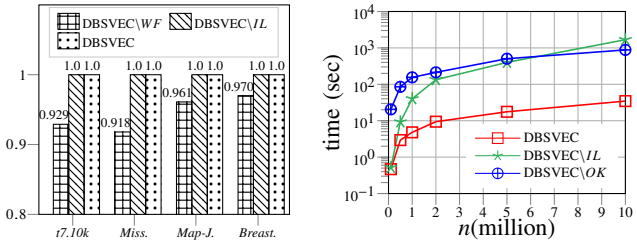
(a) Synthetic Datasets (b) *PAMAP2* (c) *Sensors* (d) *Corel-Image*

Fig. 7: Effect of radius ϵ



(a) Synthetic Datasets (n=2m) (b) Real-World Datasets

Fig. 8: Effect of penalty factor ν



(a) For improving recall (b) For improving efficiency

Fig. 9: Effect of improving

C. Computational Efficiency

For the efficiency tests, we use larger datasets as follows.

Synthetic datasets. We use synthetic datasets generated by a data generator [5]. We set the feature dimension d from 2 to 24, while the dataset cardinality n from 100 thousand to 10 million (defaults are 2 million points and 8 dimensions).

Real-world Datasets. We use three real-world datasets:

- *PAMAP2* is a 17-dimensional physical activity monitoring dataset with 1,050,199 data points [33].
- *Sensors* is an 11-dimensional dataset with 919,438 data points each representing the readings of 11 sensors [33].
- *Corel-Image* is a 32-dimensional dataset with features of 68,040 Corel images [39].

Following previous studies [5], [11], we normalize the data coordinates to $[0,10^5]$ in each dimension and use $MinPts$=100 and ϵ=5000 by default.

1) Effect of Cardinality n: We vary n from 100 thousand to 10 million. We show typical results of the algorithms over 8-dimensional synthetic data in Figure 6a (note the logarithmic scale). We see that the running times of R-DBSCAN and kd-DBSCAN increase drastically with n and quickly exceed the 10-hour limit. In contrast, the running time of DBSVEC only increases roughly linearly, taking less than 60 seconds for up to 10 million data points. DBSVEC outperforms ρ-Appr, DBSCAN-LSH, NQ-DBSCAN and k-MEANS consistently, and the advantage is up to two orders of magnitude.

2) Effect of Dimensionality d: Next, we vary d from 2 to 24 while the dataset cardinality is fixed at 2 million. Note that kd-DBSCAN takes too long to execute and is excluded from the results. ρ-Approximate is fast with low dimensions, but its performance deteriorates rapidly as d increases. This is because, as d increases, the tree structure (a *quad-tree*) constructed by ρ-Approximate grows exponentially (causing memory overflow at d = 24). In contrast, DBSVEC shows a linear growth pattern with d. Although DBSCAN-LSH, NQ-DBSCAN and k-MEANS are also linear to the number of dimensions, they are relatively slower. To evaluate DBSVEC on even higher dimensions, we generate a dataset of 1 million points with 100 dimensions. Even in this case, DBSVEC can complete in 2,057 seconds, while other methods cannot complete in 10 hours, or run out of memory.

3) Effect of Radius ϵ: Figure 7 shows the running time as the radius ϵ increases from 5,000 to 55,000. The running times of both R-DBSCAN and kd-DBSCAN increase since both algorithms rely on range queries, which are more expensive as the radius grows. The performance of DBSCAN-LSH degrades rapidly with increasing ϵ because the use of hashing to compute distance [11]. For ρ-Approximate, ϵ determines the accuracy of clustering because $\rho\epsilon$ is the minimum granularity of the grid. Although a larger radius can make ρ-Approximate faster, it reduces the clustering accuracy considerably. Moreover, in real data sets, the data space is usually large compared with the cluster radius ϵ, which causes the data space to be divided into massive grids and leads to a high running time (see Figure 7d). In comparison, DBSVEC does not have such limitations and can be better applied to real datasets. The efficiency of DBSVEC increases with the radius (fewer SVDD computation is needed). DBSVEC outperforms the baseline algorithms again in this set of experiments.

4) Effect of Penalty Factor ν: We inspect the effect of ν on the running time. Figure 8 shows that, as ν increases, DBSVEC takes longer to run. This is expected since a larger ν means allowing SVDD to generate more support vectors, which yields a higher clustering accuracy but also requires a higher computation cost.

5) Effect of Improving SVDD: We also evaluate the effect of three improvement techniques proposed for improving SVDD in DBSVEC. We denote DBSVEC using original SVDD without adaptive penalty weights as DBSVEC$\backslash WF$ and DBSVEC without incremental learning as DBSVEC$\backslash IL$, respectively. Figure 9a depicts the recall values of DBSVEC$\backslash WF$,

DBSVEC\$IL$ and DBSVEC on datasets used in Section V-B for clustering quality tests. The adaptive penalty weights improve the recall on these datasets by 3 to 8 percentage points while incremental learning has little impact on accuracy. Note that computing penalty weights adds little running time, while the aim of incremental learning is to improve efficiency without impinging accuracy. We also investigate the impact of the proposed kernel parameter value selection strategy by a variant where randomly selected kernel parameter values within a range of $\{\min_{i,j,i\neq j}\|\mathbf{x}_i-\mathbf{x}_j\|, \max_{i,j}\|\mathbf{x}_i-\mathbf{x}_j\|\}$ are used, which is denoted by DBSVEC\$OK$. Figure 9b shows the efficiency evaluation of DBSVEC\$IL$ and DBSVEC\$OK$ on the 8-dimensional synthetic data with 2 million points. It confirms that incremental learning and the kernel parameter value selection strategy help increase the efficiency of DBSVEC.

VI. Conclusions

We propose DBSVEC, a highly efficient algorithm for density-based clustering over large-scale and high-dimensional datasets. DBSVEC uses support vectors to reduce unnecessary range queries. It only performs range queries on the support vectors of sub-clusters to achieve almost the same effect as performing range queries on all the points. Furthermore, we improve SVDD via an adaptive penalty weight for each point, an incremental learning method, and a kernel parameter value selection strategy. These improvements make DBSVEC even more efficient and accurate. Extensive experiments on both synthetic and real-world datasets validate the accuracy and efficiency of DBSVEC, which is up to three orders of magnitude faster than DBSCAN. Compared with the state-of-the-art approximate density-based clustering method, DBSVEC is up to two orders of magnitude faster, and the clustering results of DBSVEC are more similar to those of DBSCAN.

Acknowledgment

This work is supported by Australian Research Council Discovery Project DP180102050 and University of Melbourne IRRTF grant.

References

[1] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters clusters in large spatial databases with noise," in *KDD*, 1996, pp. 226–231.

[2] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm gdbscan and its applications," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 169–194, 1998.

[3] A. Tramacere and C. Vecchio, "γ-ray DBSCAN: a clustering algorithm applied to Fermi-LAT γ-ray data," *Astronomy & Astrophysics*, vol. 549, no. 14, p. 138, 2013.

[4] R. Xu and D. C. Wunsch, "Clustering algorithms in biomedical research: A review," *IEEE Reviews in Biomedical Engineering*, vol. 2, no. 2, pp. 120–154, 2010.

[5] J. Gan and Y. Tao, "Dbscan revisited: Mis-claim, un-fixability, and approximation," in *SIGMOD*, 2015, pp. 519–530.

[6] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[7] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: An efficient and robust access method for points and rectangles," in *SIGMOD*, 1990, pp. 322–331.

[8] M. Huang and F. Bian, "A grid and density based fast spatial clustering algorithm," in *International Conference on Artificial Intelligence and Computational Intelligence*, 2009, pp. 260–263.

[9] M. de Berg, A. Gunawan, and M. Roeloffzen, "Faster dbscan and hdbscan in low-dimensional euclidean spaces," in *International Symposium on Algorithms and Computation*, 2017, pp. 25:1–25:13.

[10] Y. P. Wu, J. J. Guo, and X. J. Zhang, "A linear dbscan algorithm based on lsh," in *International Conference on Machine Learning and Cybernetics*, 2007, pp. 2608–2614.

[11] T. Li, T. Heinis, and W. Luk, "Hashing-based approximate dbscan," in *Advances in Databases and Information Systems*, 2016, pp. 31–45.

[12] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *VLDB*, 1999, pp. 518–529.

[13] G. Karypis, E.-H. Han, and V. Kumar, "Chameleon: hierarchical clustering using dynamic modeling," *Computer*, vol. 32, pp. 68–75, 1999.

[14] D. M. Tax and R. P. Duin, "Support vector domain description," *Pattern Recognition Letters*, vol. 20, no. 11, pp. 1191 – 1199, 1999.

[15] L. Bai, X. Cheng, J. Liang, H. Shen, and Y. Guo, "Fast density clustering strategies based on the k-means algorithm," *Pattern Recognition*, vol. 71, pp. 375 – 386, 2017.

[16] C. Li, Z. Sun, and Y. Song, "Denclue-m: Boosting denclue algorithm by mean approximation on grids," in *WAIM*, 2003, pp. 202–213.

[17] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: Why and how you should (still) use dbscan," *TODS*, vol. 42, no. 3, pp. 19:1–19:21, 2017.

[18] S. Zhou, A. Zhou, W. Jin, Y. Fan, and W. Qian, "Fdbscan: A fast dbscan algorithm," *Journal of Software*, vol. 6, no. 11, pp. 735–744, 2000.

[19] Y. Chen, S. Tang, N. Bouguila, C. Wang, J. Du, and H. Li, "A fast clustering algorithm based on pruning unnecessary distance computations in dbscan for high-dimensional data," *Pattern Recognition*, vol. 83, pp. 375 – 387, 2018.

[20] R. Zhang, B. C. Ooi, and K.-L. Tan, "Making the pyramid technique robust to query types and workloads," in *ICDE*, 2004, pp. 313–324.

[21] T. Li, T. Heinis, and W. Luk, "Advance - efficient and scalable approximate density-based clustering based on hashing," *Informatica, Lith. Acad. Sci.*, vol. 28, pp. 105–130, 2017.

[22] A. Lulli, M. Dell'Amico, P. Michiardi, and L. Ricci, "Ng-dbscan: Scalable density-based clustering for arbitrary data," *PVLDB*, vol. 10, no. 3, pp. 157–168, 2016.

[23] J. Qi, Y. Tao, Y. Chang, and R. Zhang, "Theoretically optimal and empirically efficient r-trees with strong parallelizability," *PVLDB*, pp. 621–634, 2018.

[24] X. Yang, Q. Song, and A. Cao, "Weighted support vector machine for data classification," in *IJCNN*, vol. 2, 2005, pp. 859–864.

[25] D. Kakde, A. Chaudhuri, S. Kong, M. Jahja, H. Jiang, and J. Silva, "Peak criterion for choosing gaussian kernel bandwidth in support vector data description," *CoRR*, vol. abs/1602.05257, 2016.

[26] M. Eigensatz, "Insights into the geometry of the gaussian kernel and an application in geometric modeling," Master's thesis, Swiss Federal Institute of Technology Zürich, 2006.

[27] H. Q. Minh, P. Niyogi, and Y. Yao, "Mercer's theorem, feature maps, and smoothing," in *COLT*, 2006, pp. 154–168.

[28] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[29] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett, "New support vector algorithms," *Neural Computation*, vol. 12, no. 5, pp. 1207–1245, 2000.

[30] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in kernel methods*, 1999, pp. 185–208.

[31] J. Gan and Y. Tao, "On the hardness and approximation of euclidean dbscan," *TODS*, vol. 42, no. 3, pp. 14:1–14:45, 2017.

[32] J. A. Hartigan and M. A. Wong, "A k-means clustering algorithm," *Journal of the Royal Statistical Society.*, vol. 28, no. 1, pp. 100–108, 1979.

[33] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[34] P. Fränti, O. Virmajoki, and V. Hautamäki, "Fast agglomerative clustering using a k-nearest neighbor graph," *TPAMI*, vol. 28, no. 11, pp. 1875–1881, 2006.

[35] C.J.Veenman, M.J.T.Reinders, and E.Backer, "A maximum variance cluster algorithm," *TPAMI*, vol. 24, no. 9, pp. 1273–1280, 2002.

[36] P. Fränti, M. Rezaei, and Q. Zhao, "Centroid index: cluster level similarity measure," *Pattern Recognition*, vol. 47, pp. 3034–3045, 2014.

[37] P. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, no. 1, pp. 53–65, 1987.

[38] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 1, no. 2, pp. 224–227, 1979.

[39] K. Chakrabarti and S. Mehrotra, "The hybrid tree: An index structure for high dimensional feature spaces," in *ICDE*, 1999, pp. 440–447.